Implementing the finite element method with libfemtools

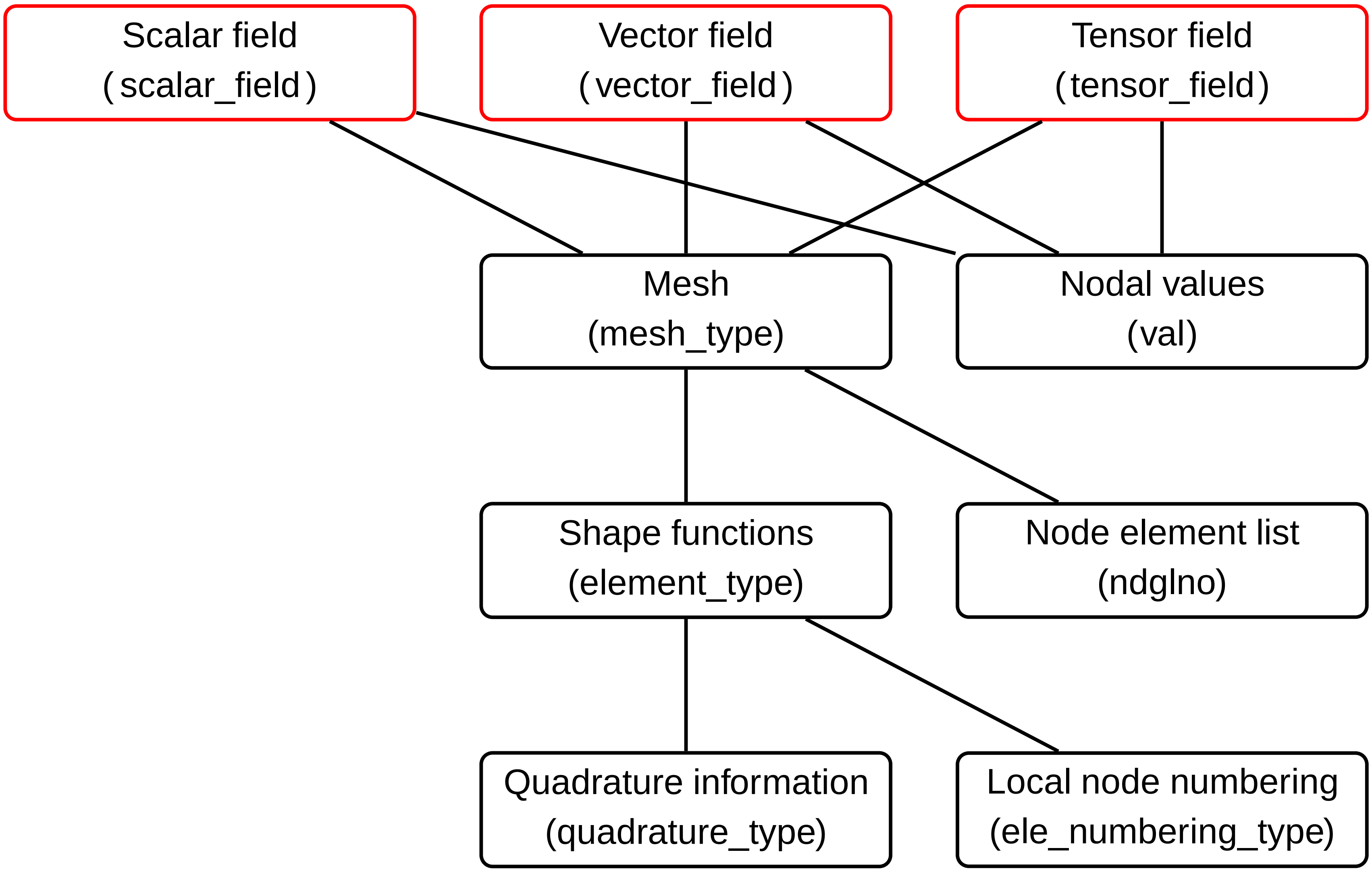
Computer techniques for modellers

David Ham

`David.Ham@imperial.ac.uk`

Imperial College London

Field type heirarchy



Imperial College
London

Field objects

An abbreviated version of the scalar field definition:

```fortran
type scalar_field
   !! Field value at points.
   real, dimension(:), pointer :: val
   !! Flag for whether val is allocated
   logical :: wrapped=.true.
   type(scalar_boundary_condition), dimension(:), pointer :: &
        boundary_condition => null()
   character(len=FIELD_NAME_LEN) :: name=""
   !! path to options in the options tree
   character(len=OPTION_PATH_LEN) :: option_path=""
   type(mesh_type) :: mesh
   !! Reference count for field
   type(refcount_type), pointer :: refcount=>null()
   !! Indicator for whether this is an alias to another field.
   logical :: aliased=.false.
end type scalar_field
```

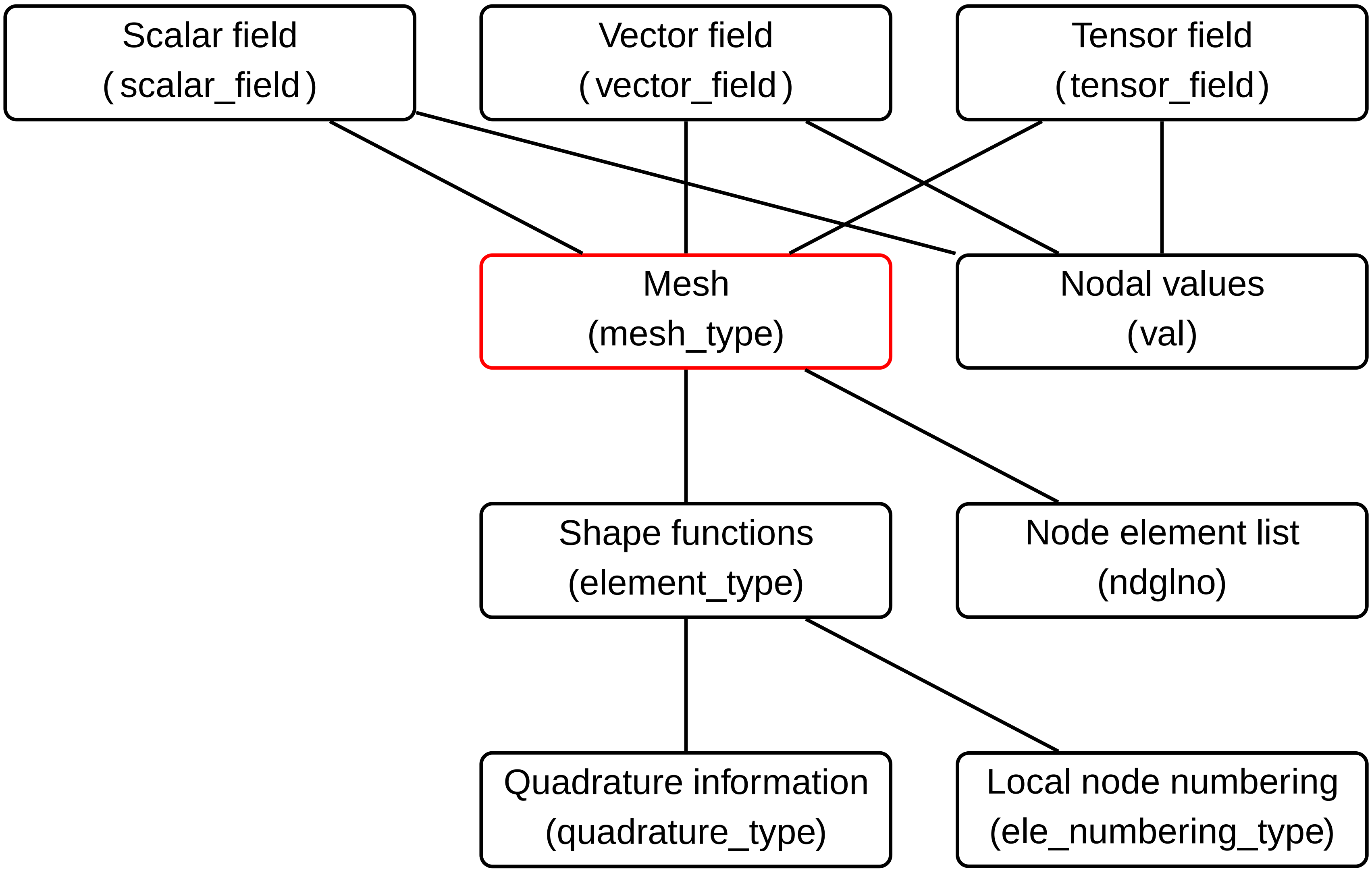
Note that for most purposes fields are *opaque* types.

Field methods

```fortran
use fields
type(scalar_field) :: temperature
type(mesh_type) :: model_mesh
real, dimension(:), pointer :: t_ele

call allocate(temperature, model_mesh)
! Set the whole field to 0.0
call zero(temperature)
! Find the nodes of the third element.
t_ele=>ele_nodes(temperature, 3)
! Values at nodes of third element:
print *, ele_val(temperature, 3)
! Values at quadrature points in third element:
print *, ele_val_at_quad(temperature, 3)
! Number of nodes in third element:
print *, ele_loc(temperature,3)
```

Field type heirarchy



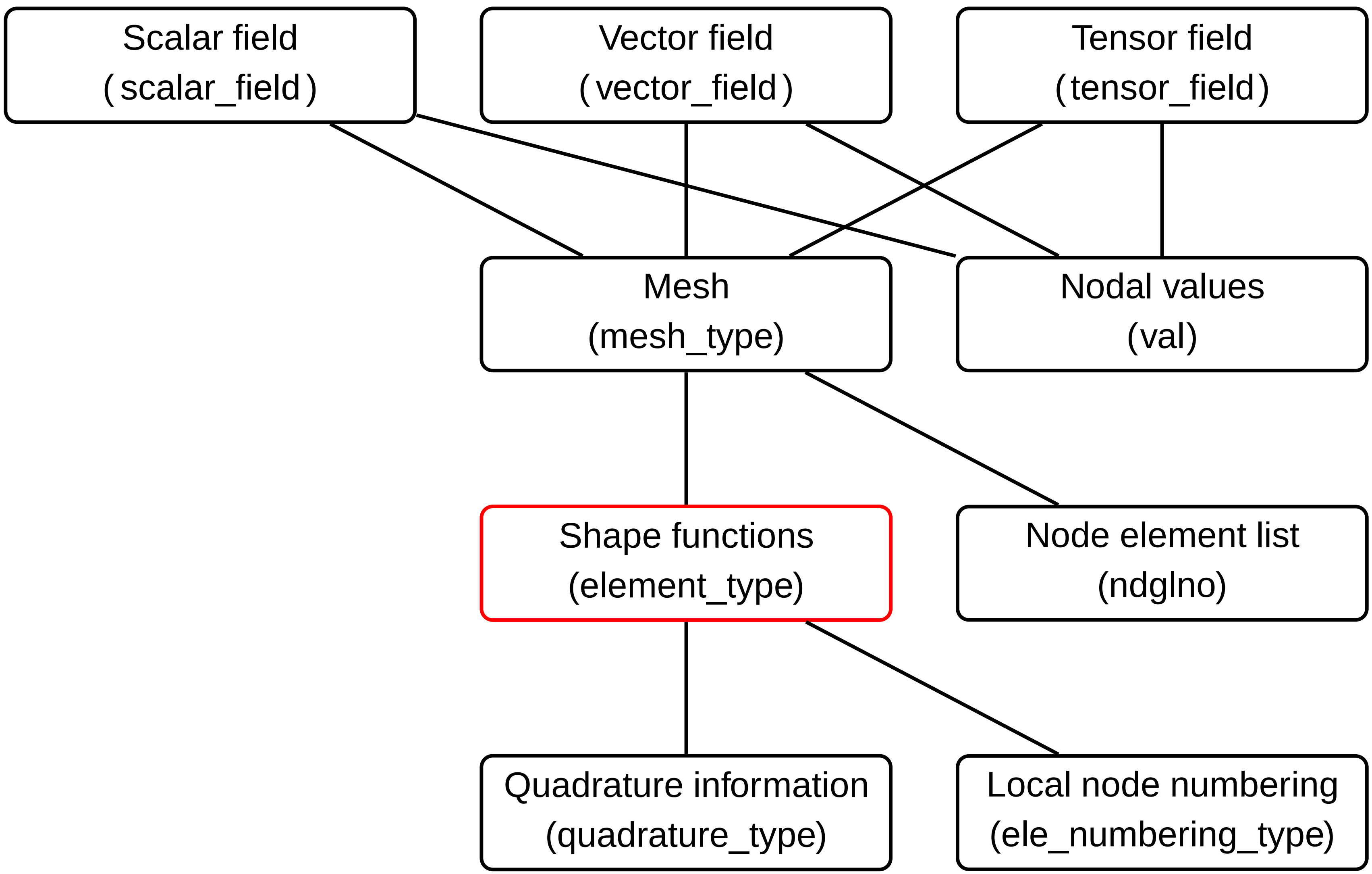
Imperial College
London

Mesh objects

Many fields can share one mesh if they have the same finite element space.

```
type mesh_type
  !!< Mesh information for (among other things) fields.
  integer, dimension(:), pointer :: ndglno
  !! Flag for whether ndglno is allocated
  logical :: wrapped=.true.
  type(element_type), pointer :: shape
  integer :: elements ! Number of elements
  integer :: nodes ! Number of nodes
  character(len=FIELD_NAME_LEN) :: name
  ! Degree of continuity of the field. 0 is for the conventional
  ! C0 discretisation. -1 for DG.
  integer :: continuity=0
  ! Mesh face information when needed (eg DG).
  type(mesh_faces), pointer :: faces=>null()
end type mesh_type
```

Field type heirarchy



Element shape functions

```
type element_type
   integer :: dim !! 2d or 3d?
   integer :: loc !! Number of nodes.
   integer :: ngi !! Number of gauss points.
   integer :: degree !! Polynomial degree of element.
   !! Shape functions: n is for the primitive function,
   !! dn is for partial derivatives.
   !! n is loc x ngi, dn is loc x ngi x dim
   real, pointer :: n(:,:)=>null(), dn(:,:,:)=>null()
   !! Polynomials defining shape functions and their derivatives.
   type(polynomial), dimension(:,:), pointer :: spoly=>null(), &
        &                                      dspoly=>null()
   !! Link back to the node numbering used for this element.
   type(ele_numbering_type), pointer :: numbering=>null()
   !! Link back to the quadrature used for this element.
   type(quadrature_type), pointer :: quadrature=>null()
end type element_type
```

Imperial College
London

Making quadrature and elements

```
type(quadrature_type) :: quad
type(element_type) :: shape

! Make a triangular element.
quad=make_quadrature(loc=3, dimension=2, degree=3)

shape=make_element_shape(loc=3, dimension=2, degree=1, quad=quad)

! Quadrature and elements are dynamically sized and must be
! deallocated when no longer needed.
call deallocate(quad)
call deallocate(shape)
```

Imperial College London

State objects

A state object stores fields and meshes by *name*:

```
use state_module
use fields

type(vector_field) :: position
type(scalar_field) :: temperature
type(tensor_field) :: temp_diffusivity

type(state_type) :: state

call insert(state, position, 'Coordinate')
call insert(state, temperature, 'Temperature')
call insert(state, temp_diffusivity, 'TemperatureDiffusivity')
call insert(state, position%mesh, 'Coordinate Mesh')
```

Any number of fields can be stored in this way.

State objects

Retrieval functions from state objects return *pointers* not copies:

```
subroutine state_operation(state)
  use state_module
  use fields
  type(scalar_field), pointer :: temperature
  type(tensor_field), pointer :: temp_diffusivity
  type(mesh_type), pointer :: X_mesh
  integer :: stat

  temperature=>extract_scalar_field(state, 'Temperature', &
       &                               stat=stat)
  ! If no temperature then do nothing.
  if (stat/=0) return
  ...

end subroutine state_operation
```

Imperial College London

An equation!

$$\nabla^2 \psi = f(\mathbf{x})$$

on some domain Ω with boundary condition $\nabla\psi \cdot \mathbf{n} = 0$.

An equation!

$$\nabla^2\psi = f(\mathbf{x})$$

on some domain Ω with boundary condition $\nabla\psi \cdot \mathbf{n} = 0$.

Multiply by a test function and integrate:

$$\int_\Omega N\nabla^2\psi dV = \int_\Omega N f(\mathbf{x}) dV$$

An equation!

$$\nabla^2 \psi = f(\mathbf{x})$$

on some domain Ω with boundary condition $\nabla\psi \cdot \mathbf{n} = 0$.

Multiply by a test function and integrate:

$$\int_\Omega N \nabla^2 \psi dV = \int_\Omega N f(\mathbf{x}) dV$$

Integrate by parts.

$$-\int_\Omega \nabla N \nabla \psi dV + \int_\Gamma N \nabla \psi \cdot \mathbf{n} dA = \int_\Omega N f(\mathbf{x}) dV$$

An equation!

$$\nabla^2 \psi = f(\mathbf{x})$$

on some domain Ω with boundary condition $\nabla\psi \cdot \mathbf{n} = 0$.

Multiply by a test function and integrate:

$$\int_\Omega N \nabla^2 \psi dV = \int_\Omega N f(\mathbf{x}) dV$$

Integrate by parts.

$$-\int_\Omega \nabla N \nabla \psi dV + \int_\Gamma N \nabla \psi \cdot \mathbf{n} dA = \int_\Omega N f(\mathbf{x}) dV$$

An equation!

$$\nabla^2 \psi = f(\mathbf{x})$$

on some domain Ω with boundary condition $\nabla\psi \cdot \mathbf{n} = 0$.

Multiply by a test function and integrate:

$$\int_\Omega N \nabla^2 \psi dV = \int_\Omega N f(\mathbf{x}) dV$$

Integrate by parts.

$$-\int_\Omega \nabla N \nabla \psi dV = \int_\Omega N f(\mathbf{x}) dV$$

A test case

As a simple test case we take:

$$f(x,y) = -4.0\pi^2 \mathrm{dim} \prod_{i=1}^{\mathrm{dim}} \cos(2\pi \mathbf{x}_i)$$

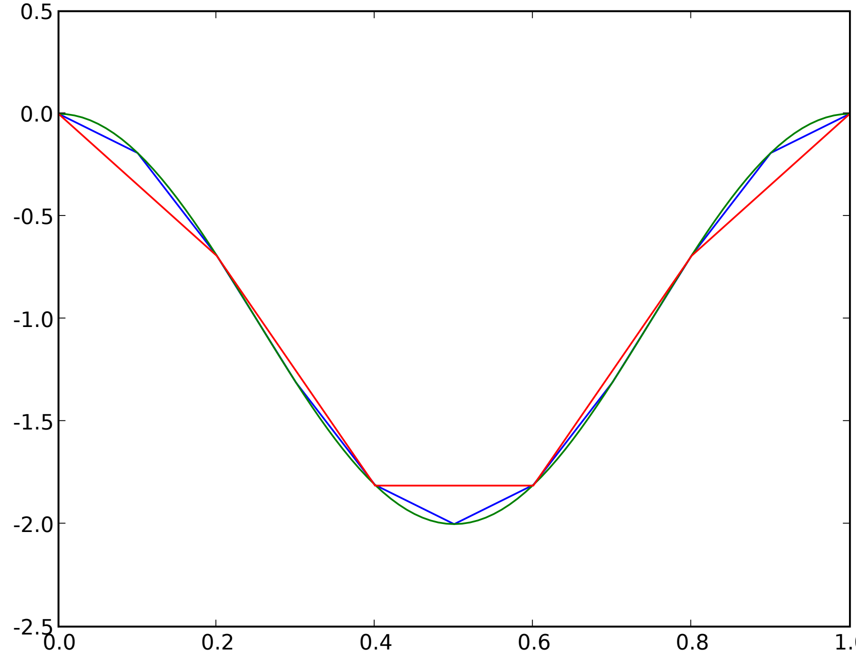
Which has analytic solution:

$$\Psi(x,y) = \prod_{i=1}^{\mathrm{dim}} \cos(2\pi \mathbf{x}_i) + C$$

for an arbitrary constant C.

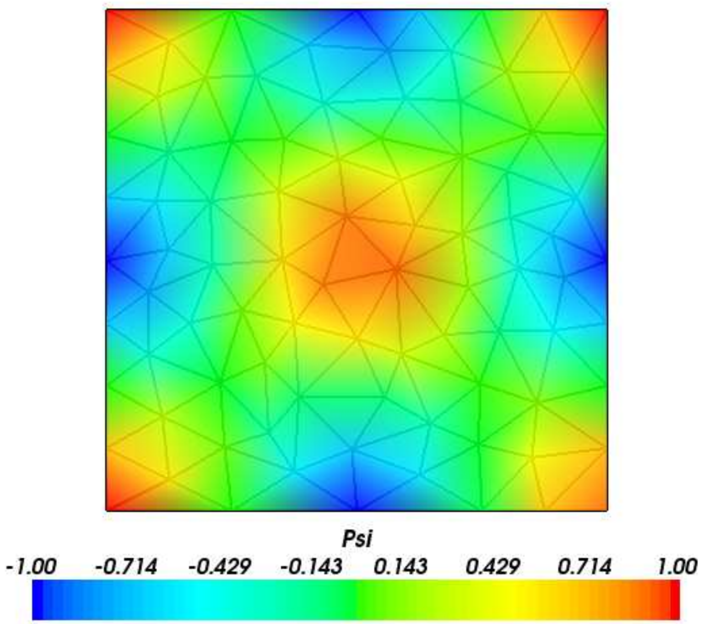
Go read some source

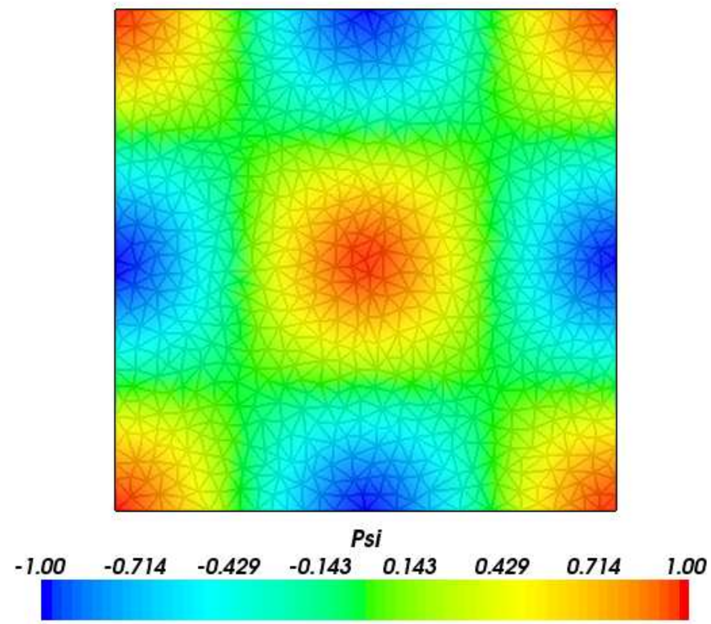
1D results



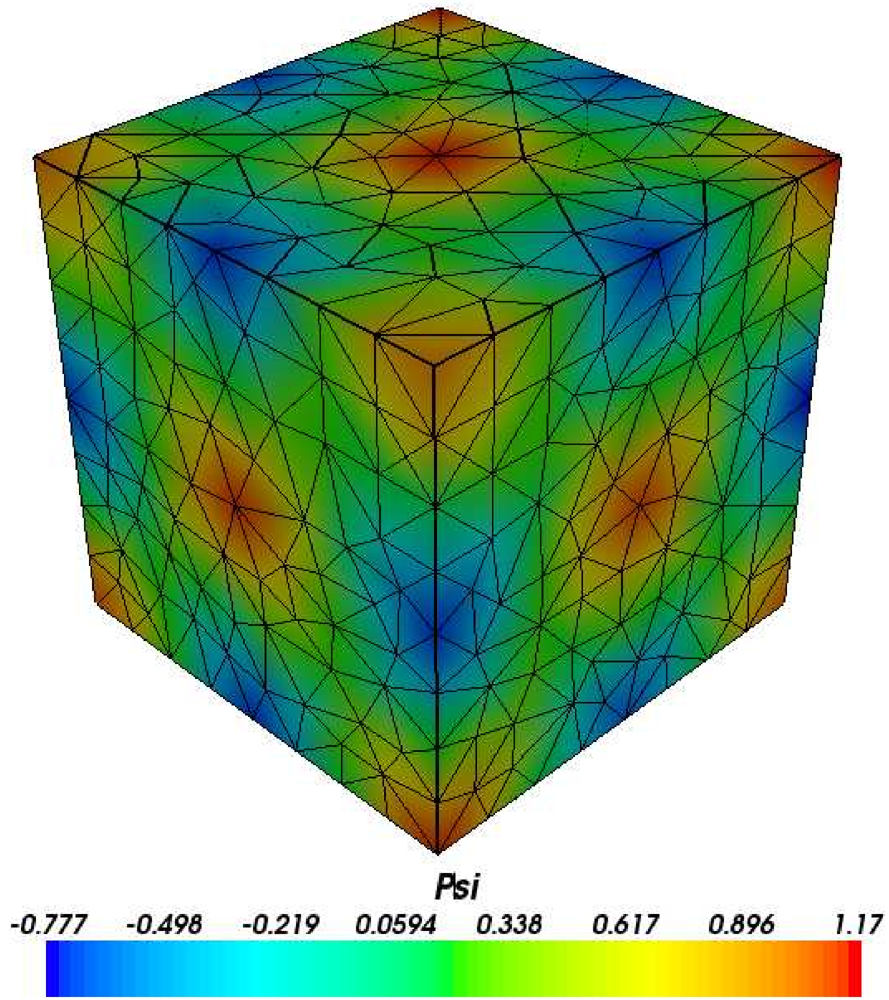
Imperial College London

2D results


Psi
-1.00
-0.714
-0.429
-0.143
0.143
0.429
0.714
1.00

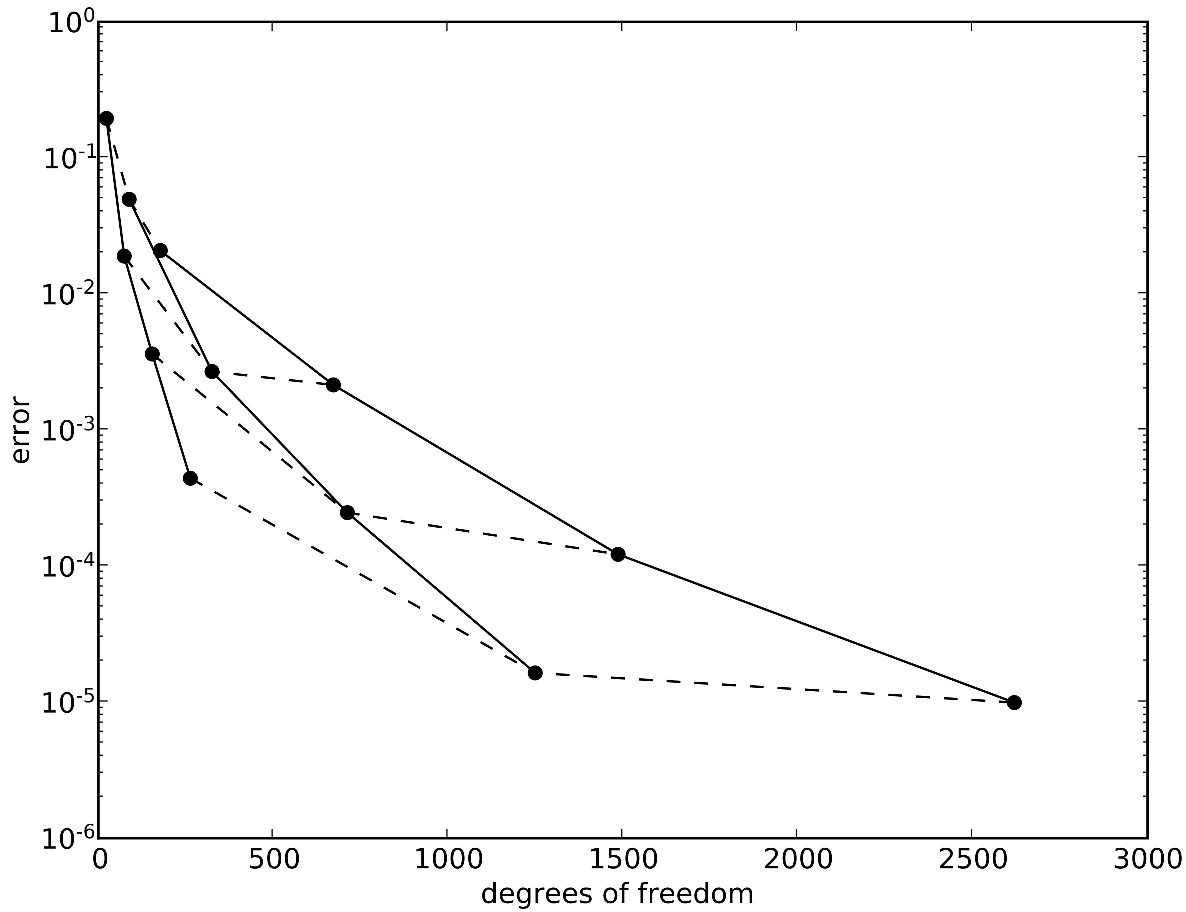

Psi
-1.00
-0.714
-0.429
-0.143
0.143
0.429
0.714
1.00

3D results



Imperial College London

Error for h and p refinement



Error against degrees of freedom. Solid lines link results on the same mesh, broken lines link results with the same degree elements

Imperial College London